***OxDEV Hackathon Project Document***

Problem Statement:

"DreamCanvas: AI-Powered Creative Expression for Children with Motor Disabilities"

Children with motor disabilities often face significant barriers in expressing their creativity through traditional art-making methods. Many existing digital art tools require precise motor control or complex interface navigation, limiting these children's ability to participate in creative activities that are crucial for cognitive development and emotional expression. Basically, we will be developing an AI-powered generative art platform that enables children with motor disabilities to create complex artwork using simple, accessible inputs (like voice commands, eye tracking, or single-switch controls).

**Project Name:** DreamCanvas  
**Objective:** Empower children with motor disabilities to express creativity through an AI-powered generative art platform.

**Technical Structure and Workflow**

**1. Frontend Development**

**Technologies:**

* React.js
* TypeScript
* Tailwind CSS
* WebGazer.js (Eye-tracking)
* ARIA compliance tools

**Workflow:**

1. **Set Up React Project**:
   * Use Vite for project scaffolding.
   * Install necessary dependencies: React, TypeScript, Tailwind CSS.
   * Configure Tailwind CSS for styling.
2. **Build the UI**:
   * Design mockups for the homepage, art canvas, and settings.
   * Implement responsive and adaptive layouts using Tailwind CSS.
   * Add ARIA roles and attributes for screen reader compatibility.
3. **Accessibility Features**:
   * Integrate WebGazer.js for eye-tracking functionality:
     + Capture gaze points and map them to cursor movement.
     + Optimize calibration for accuracy.
   * Implement custom keyboard navigation controls.
   * Add voice command integration using a JavaScript voice recognition library (e.g., annyang.js).
4. **Connect with Backend**:
   * Use RESTful APIs to send user commands to the backend and receive generative art data.
   * Integrate real-time updates via WebSockets (e.g., socket.io).

**2. Backend Development**

**Technologies:**

* Flask (Python)
* Node.js (for WebSocket integration)
* SQLite/PostgreSQL (Database)

**Workflow:**

1. **Set Up Flask Project**:
   * Initialize the Flask application.
   * Define routes for API endpoints (e.g., for voice command processing, eye-tracking data, and artwork generation).
2. **Stable Diffusion API Integration**:
   * Use a Python wrapper (like diffusers) for the Stable Diffusion model.
   * Create endpoints for:
     + Accepting user inputs (voice commands, eye-tracking data).
     + Generating and returning artwork based on these inputs.
3. **Data Handling**:
   * Store user session data and preferences in a database (SQLite/PostgreSQL).
   * Design schemas to support user profiles, accessibility settings, and generated artworks.
4. **Real-Time Feedback**:
   * Integrate WebSocket functionality using Flask-SocketIO or Node.js.
   * Push updates to the frontend as the artwork is being generated.

**3. Integration of Accessibility Features**

**Technologies:**

* WebGazer.js
* ARIA roles
* Voice Recognition API (e.g., annyang.js)

**Workflow:**

1. **Eye-Tracking Calibration**:
   * Set up an interactive calibration interface.
   * Process gaze data to enable precise interaction with the canvas.
2. **Voice Commands**:
   * Define a dictionary of voice commands (e.g., “Start Drawing”, “Change Color”).
   * Link voice commands to UI actions and API requests.
3. **Single-Switch Controls**:
   * Implement navigation and action triggers using keyboard or external switches.
   * Support dynamic menu configurations based on user needs.

**4. AI-Powered Art Generation**

**Technologies:**

* Stable Diffusion API (Open Source)
* Hugging Face Transformers

**Workflow:**

1. **Prepare the Model**:
   * Download and set up the Stable Diffusion model using diffusers.
   * Optimize inference for real-time usage.
2. **Input Processing**:
   * Map user inputs (e.g., voice commands, gaze points) to Stable Diffusion parameters.
   * Allow customization of artistic styles and color palettes.
3. **Output Delivery**:
   * Save generated artworks temporarily on the server.
   * Send artwork data to the frontend for display.

**5. Deployment**

**Technologies:**

* Docker
* Open Source Hosting (e.g., Railway.app, Render.com, or Heroku)

**Workflow:**

1. **Containerize the Application**:
   * Create Dockerfiles for both frontend and backend.
   * Use Docker Compose for multi-service orchestration.
2. **Deploy to Hosting Platform**:
   * Push the containerized application to an open-source hosting platform.
   * Configure environment variables and persistent storage for the database.
3. **Open Source Documentation**:
   * Set up a GitHub repository with a clear README.
   * Include instructions for local setup and contributions.

**6. Future Enhancements**

1. **Progressive Learning Paths**:
   * Introduce tutorials and challenges to help users explore art styles and tools.
2. **Community Features**:
   * Allow users to share their artwork in a community gallery.
   * Enable feedback and collaboration on creative projects.
3. **Offline Support**:
   * Develop a lightweight offline version for limited internet access.

**Action Plan**

1. Target 1:
   * Set up frontend and backend scaffolding.
   * Implement basic UI and Flask API endpoints.
2. **Target 2**:
   * Integrate accessibility features (eye-tracking, voice commands).
   * Implement Stable Diffusion API integration.
3. **Target 3**:
   * Develop real-time feedback mechanism.
   * Test and optimize input-output flow.
4. **Target 4**:
   * Finalize deployment.
   * Document the project and prepare a demo.